DSR 路由协议代码分析

学号	姓名	班级	负责模块	成绩
201692065	王哲	软网 1601	任务的分工，工作总结，路由协议过程理解	
201692348	吴珺益	软网 1601	文件分析，数据结构	

目录

引　言

动态源路由协议(Dynamic Source Routing Protocol,DSR)是一个专门为多跳无线 Ad Hoc 网络设计的简单且高效的路由协议。所有的路由都是由 DSR 路由协议动态地、自动地确定和维护，它提供快速反应式服务，以便帮助确保数据分组的成功交付，即使在节点移动或者其他网络状况变化的条件下也是如此。本文档将根据 DSR 协议的源代码分成协议原理，代码介绍，数据结构及具体算法实现四个部分进行详细的介绍。

第一章 协议介绍

动态源路由协议 DSR （Dynamic Source Routing）

DSR 的特点在于使用了源路由的路由机制，在每一个分组的头部都携带整条路由的信息，路由器按照该路由纪录来转发分组。这种机制最初被 IEEE802.5 协议用在由桥互连的多个令牌环网中寻找路由。

DSR 借鉴该机制，并结合了按需路由的思想。DSR 协议使用源路由，采用 Cache（缓冲器）存放路

由信息，且中间节点不必存储转发分组所需的路由信息，网络开销较少，但存在陈旧路由。

Dynamic Source Routing 按需路由

节点需要发送数据时才进行路由发现过程

反应型路由，仅维护活跃的路由

源路由

发送节点在分组中携带到达目的节点的路由信息（转发分组的完整的节点序列） - 不需要中间节点维护路由息

节点缓存到目的节点的多条路由 - 避免了在每次路由中断时都需要进行路由发现，因此能够对拓扑变化作出更快的反应。

路由发现（Route Discovery）

只有在源节点需要发送数据时才启动

帮助源节点获得到达目的节点的路由

路由维护（Route Maintenance）

在源节点在给目的节点发送数据时监测当前路由的可用情况

当网络拓扑变化导致路由故障时切换到另一条路由或者重新发起路由发现过程

路由发现和路由维护都是按需进行的

不需要周期性路由公告

不需要感知链路状态

不需要邻居检测

DSR 协议操作

（1）路由发现

当一个节点欲发送数据到目的节点时，它首先查询路由缓冲器是否有到目的节点的路由。如果

有，则按此路由发送数据；如果没有，源节点就开始启动路由发现程序。路由发现过程中使用

洪泛路由（Flooding Routing）。

路由发现的具体处理过程：

当节点 S 需要向节点 D 发送数据，但不知到节点 D 的路由，于是节点 S 就开始路由发现过程。

源节点 S 洪泛“路由请求”分组 Route Request (RREQ)，每个请求分组通过序列号和源节点 S 标识唯一确定。

DSR 路由发现：路由请求

源节点向邻居节点广播路由请求（RREQ：Route Request）目的节点地址

路由记录：纪录从源节点到目的节点路由中的中间节点请求 ID

中间节点接收到 RREQ 后，将自己的地址附在路由纪录中

DSR 路由发现：中间节点处理

中间节点维护<源节点地址、请求 ID>序列对列表

重复 RREQ 检测

如果接收到的 RREQ 消息中的<源节点地址、请求 ID>存在于本节点的序列对列表中

如果接收到的 RREQ 消息中的路由纪录中包含本节点的地址

如果检测到重复，则中间节点丢弃该 RREQ 消息

DSR 路由发现：路由应答

目的节点收到 RREQ 后，给源节点返回路由应答（RREPRoute Reply）消息

拷贝 RREQ 消息中的路由纪录

源节点收到 RREP 后在本地路由缓存中缓存路由信息。

DSR 路由维护

逐跳证实机制

链路层

确认

被动确认（监听其它节点间的数据发送）

其它高层

要求 DSR 软件返回确认

端到端证实机制

无法确定故障发生的位置

DSR 逐跳证实机制

DSR 路由维护

如果数据分组被重发了最大次数仍然没有收到下一跳的确认，则节点向源端发送路由错误（Route Error）消息，并且指明中断的链路

源端将该路由从路由缓存中删除

如果源端路由缓存中存在另一条到目的节点的路由则使用该路由重发分组

否则重新开始路由发现过程

DSR 路由应答（链路为双向的）：

（1）当目的节点 D 一接到 RREQ 分组，就发送 RREP 分组

（2）RREP 分组中包含有 RREQ 分组中从源节点 S 到目的节点的路由纪录（前向路）

（3）RREP 分组按 RREQ 分组的路由纪录进行反向传送。

DSR 路由应答（链路为单向的）：

此时，目的节点执行和源节点相同的路由发现过程，

所不同的的是，目的节点的 RREQ 分组捎带传送 RREP 分组。

DSR 的路由缓存

（1）当源节点 S 接到 RREP 分组后，就将 RREP 分组中从源节点 S 到目的节点 D 的路由信息进行缓存

（2）当源节点 S 向目的节点 D 发送数据分组时，此路由信息就包含在每个分组的头部。

（3）所有的中间节点利用源路由信息进行分组转发。

DSR 协议特点

1）节点不需要周期性的发送路由广播分组，无需维护去往全网所有节点的路由信息，能自然而然的消除路由环路，而且能提供多条路由，可用于单向信道；

2）支持中间节点的应答，能使源节点快速获得路由，但会引起过时路由问题；

3）每个分组都需要携带完整的路由信息，造成开销较大，降低了网络带宽的利用率，不适合网络直径大的自组网，网络扩展性不强；

DSR 优化：路由缓存

每个节点缓存它通过任何方式获得的新路由

转发 RREQ

获得从本节点到 RREQ 路由记录中所有节点的路由，例如 E 转发 RREQ(A-B-C)获得到到 A 的路由(C-B-A)

转发 RREP

获得本节点到 RREP 路由纪录中所有节点的路由，例如 B 转发 RREP(A-B-C-D)获得到 D 的路由(C-D)

转发数据分组

获得从本节点到数据分组节点列表中所有节点的路由，例如 E 转发数据分组(A-B-C)获得到 A 的路由(C-B-A)

监听相邻节点发送的分组

RREQ、RREP、数据分组等

中间节点使用缓存的到目的节点的路由响应 RREQ

RREP 中的路由纪录=RREQ 中的路由纪录+缓存的到目的节点的路由

错误路由缓存

网络拓扑的变化使得缓存的路由失效

影响和感染其它节点，使用该路由缓存的路由将不可用

当节点根据路由缓存回应 RREP 时，其它监听到此 RREP 的节点会更改自己缓存的路由，从而感染错误路由缓存

设置缓存路由的有效期，过期即删除

RREP 风暴

节点广播到某个目的节点的 RREQ，当其邻居节点的路由缓存中都有到该目的节点的路由时，每个邻居节点都试图以自己缓存的路由响应，由此造成 RREP 风暴

RREP 风暴将浪费网络带宽，并且加剧局部网络冲突

预防 RREP 风暴

每个节点延时 D 发送 RREP

D=H *（h-1+r）

其中 H 是每条链路的传播延时 h 是自己返回的路径长度，即到目的节点的跳数 r 是 0 或者 1

D 与节点到目的节点的跳数成正比，使得到目的节点有最短路径的 RREP 最先发送

节点将接口设置成混杂模式(promiscuous)，监听是否存在有比自己更短的到目的节点的路径，如果有，则不发送本节点的 RREP

DSR 协议的优缺点

优点:

采用源路由机制、避免了路由环路。

它是一种按需路由协议、只有当两个节点间进行通信时，才会缓存路由纪录，因此相对主动路由来说，减小了路由维护的开销。

通过采用路由缓存技术，减少路由请求信息对信道的占用

缺点:

随着路经跳数的增加，分组头长度线性增加、开销大

路由请求分组 RREQ 采用洪泛发向全网扩散，导致网络负荷大

来自邻居节点的 RREQ 分组在某个节点可能会发生碰撞，解决办法是：在发送 RREQ 分组时引入随机时延

当源节点发送路由请求分组 RREQ 时，可能会收到多个节点缓存的到达目的节点的路由信息，引起竞争。解决办法：若某节点听到其它节点发出的 RREQ 分组中路由信息含有较少跳数，此节点停止发送。

当源节点发送路由请求分组 RREQ 时，可能会收到多个节点缓存的到达目的节点的路由信息，但有些路由信息可能是过时的。解决办法：引入定时器、链路断的情况应进行全网洪泛。

第二章 代码介绍

2.1 文件介绍

我们发现 DSR 协议中有很多源文件，下表就是我们在遇到这些源文件时找到的描述，我们选择比较重要的部分描述

文件	描述
Dsr_ack.c	用于处理接收回复确认
Dsr_ack.h	定义 req 和 ack
Dsr_rrep.c	用来处理 rrep 包裹的各种事务
Dsr_rreq.c	用来处理 rreq 包裹的各种事务
Dsr_rreq.h	定义 rreq 包裹的结构
Dsr_rrep.h	定义 rrep 包裹的结构
Dsr_rrer.c	路由错误消息；路由错误分组；向中心节点发送路由错误帧
Dsr_pkt.h	对包的长度类型等进行了规定
Neigh.c	负责更新和存储路由表

第二章 数据结构

1.路由缓存

实现 DSR 的每个节点必须维护一个包含的路由缓存节点所需的路由信息 。节点添加信息它的路由缓存，因为它了解 ad hoc 中节点之间的新链接网络; 例如，节点可以在收到时了解新链接携带路由请求，路由回复或 DSR 源路由的数据包。同样，节点会从其路由缓存中删除信息了解到 ad hoc 网络中的现有链路已经崩溃。对于例如，节点在接收到数据包时可能会知道链路断开携带路由错误或通过链路层重传报告将数据包转发到其下一跳的失败的机制目的地。

任何时候节点向其路由缓存（节点）添加新信息应该检查自己的发送缓冲区中的每个数据包现在确定是否有到该数据包的 IP 目的地地址的路由存在于节点的路由缓存中（包括信息）添加到缓存中）。如果是这样，那么应该使用发送数据包该路由并从发送缓冲区中删除。

可以将 DSR 网络与其他网络连接，此 DSR 网络外部。这样的外部网络可以用于例如，可以是因特网，也可以是其他 ad hoc 网络 DSR 以外的路由协议。这样的外部网络也可以是按顺序视为外部网络的其他 DSR 网络提高可扩展性。完全处理这种外部网络超出了本文档的范围。但是，这个 document 指定了一组最低要求和功能

必须允许节点仅实现此规范与实现此类接口的节点正确地互操作外部网络。这一小组要求和功能涉及第一跳外部（F）和最后一跳外部（L）位 DSR 源路由选项和路由回复选项数据包的 DSR 选项标题中的。这些要求还包括添加外部标志位标记路由缓存中的每个链接，从第一跳复制 DSR 源路由中的外部（F）和最后一跳外部（L）位选项或路由回复选项，从中学习此链接。

路由缓存应该支持为每个路由存储多个路由目的地。在路由缓存中搜索到某些路由目标节点，目标节点搜索路由缓存地址。以下属性描述了此搜索功能在路由缓存上：

- 任何节点上的 DSR 的每个实现都可以选择任何适当的搜索路由缓存和选择的策略和算法

从找到的目的地到目的地的“最佳”路线。对于例如，节点可以选择选择到达的最短路径目标（最短的跳跃序列），或者它可以使用备用度量从 Cache 中选择路由。

- 但是，如果有多个到目的地的缓存路由，则搜索路由缓存时选择的路由应该更喜欢没有在任何链接上设置外部标志的路由。这个 preference 将选择直接指向目标的路径尝试通过任何外部目标到达目标的路由上的节点连接到 DSR ad hoc 网络的网络。
- 此外，搜索路由缓存时选择的任何路由不得为除以外的任何链接设置外部位可能是第一个链接，最后一个链接，或两者; 外部位不得为所选路由中的任何中间跃点设置。

路由缓存的实现可以提供固定容量缓存或缓存大小可以是可变的。下列属性描述节点内可用空间的管理路由缓存：

- 每个节点的 DSR 的每个实现可以选择任何适当的用于管理其路由缓存中的条目的策略，例如何时有限的缓存容量需要选择哪些条目保留在缓存中。例如，节点可以选择“最少最近使用了“（LRU）缓存替换策略，其中的条目如果做出决定，最后使用的最后一次被从缓存中丢弃需要在高速缓存中允许一些新条目的空间被添加。
- 但是，路由缓存替换策略应该允许路由到根据“偏好”进行分类，其中路线较高偏好不太可能从缓存中删除。对于例如，节点可能更喜欢它发起路由的路由发现它是滥交后学到的路线窥探其他数据包。特别是，节点应该更喜欢目前使用的路线不是那些路线。

任何合适的数据结构组织，与此一致规范，可用于在任何节点中实现路由缓存。例如，以下两种类型的组织是可能的：

- 在 DSR 中，在收到的每个 Route Reply 中返回的路由路线发现的发起者（或从路上学到的）开销数据包的标头，表示通向的完整路径（一系列链接）目标节点。通过分别缓存这些路径中的每一个，a 可以形成路由缓存的“路径缓存”组织。一个路径缓存实现起来非常简单，并且很容易保证所有路线都是无环路的，因为每条路线都来自一条路线回复或路由请求或在数据包中使用是无环路的。至在路径缓存数据结构中搜索路由，发送节点可以简单地在其路由缓存中搜索任何路径（或者前缀）通向预期目标节点的路径。

- 或者，可以使用“链接缓存”组织路由缓存，其中路由中的每个单独链路（跳）在路由回

复包中返回（或以其他方式从中学习开销数据包的标题）被添加到统一的图数据中此节点的结构当前网络拓扑视图。至在链接缓存中搜索路由，发送节点必须使用更多复杂的图搜索算法，如着名的 Dijkstra's 最短路径算法，找到当前最佳路径图表到目标节点。这样的算法更多难以实现，可能需要更多的 CPU 时间执行。

但是，链接缓存组织比路径更强大缓存组织，在其有效利用所有的能力节点可能了解的有关状态的潜在信息的网络。特别是从不同的路线学到的链接发现或从任何听到的数据包的标题可以是合并在一起形成网络中的新路由，但事实并非如此由于每个人的分离，可能在路径缓存中缓存中的路径。

用于路由缓存的数据结构组织的选择在任何 DSR 实现中，每个节点的本地事务都会受到影响只有表现; 路线的任何合理的组织选择缓存不会影响正确性或互操作性。

路由缓存中的每个条目都应该具有与之关联的超时它，如果在一段时间内没有使用该条目，则允许删除该条目。算法和数据结构的特殊选择用于实现路由缓存应该在选择时考虑路由缓存中的条目超时。配置变量定义的 RouteCacheTimeout 指定了超时应用于路由缓存中的条目，尽管也可以而是使用自适应策略来选择超时值而不是对所有条目使用单个超时设置。例如，Link-MaxLife 缓存设计使用自适应超时算法并不使用 RouteCacheTimeout 配置变量。

Link-MaxLife 是一个自适应链接缓存，其中包含每个链接缓存具有由缓存动态确定的超时节点根据其观察到的两个节点的过去行为链接的末尾。另外，在为数据包选择路由时被发送到某个目的地，在相同长度的缓存路线中到目的地的（跳数），Link-MaxLife 选择路线具有最长的预期寿命（任何最高的最小超时）链接在路线中）。使用 Link-MaxLife 设计进行路线在 DSR 的实现中建议使用高速缓存。

2.发送缓冲区

实现 DSR 的节点的发送缓冲区是一个包的队列无法由该节点发送，因为它还没有源路由到每个这样的数据包的目的地。发送中的每个数据包缓冲区在逻辑上与放入的时间相关联缓冲区应该从发送缓冲区中以静默方式删除在最初生成后的一段 SendBufferTimeout 之后丢弃

放在缓冲区。如有必要，应该使用 FIFO 策略在超时之前驱逐数据包以阻止缓冲区四溢。

3.路线请求表

实现 DSR 记录的节点的路由请求表有关最近发起的路线请求的信息或由此节点转发。该表由 IP 地址索引。

节点上的路由请求表记录以下信息关于此节点已向其发起路由请求的节点：

- Route 的 IP 头中使用的生存时间（TTL）字段请求此节点发起的最后一次路由发现那个目标节点。该值允许节点实现 a 用于控制其路线传播的各种算法针对目标启动的每个 Route Discovery 请求。

- 此节点上次为此发起路由请求的时间目标节点。

- 为此启动的连续路由发现的数量目标，因为收到一个有效的路由回复，给出一个路由目标节点。

- 此节点可以接下来的剩余时间尝试针对该目标节点的 Route Discovery。当节点为此目标节点（此字段）启动新的路由发现在该目标节点的 Route Request Table 条目中初始化为 Route Discovery 的超时，之后节点可以为该目标启动新的发现。直到收到此目标节点地址（节点）的有效路由回复必须在确定此超时时实施退避算法

为此启动的每个连续路径发现的值目标在 IP 标头中使用相同的生存时间（TTL）值路由请求包。这种连续之间的超时路线发现启动应该通过加倍来增加每个新启动的超时值。

此外，节点上的路由请求表也记录了以下有关此节点所具有的启动器节点的信息收到路线请求：

- 包含大小的 RequestTableIds 条目的 FIFO 缓存最近路线的识别值和目标地址此节点从该启动器节点收到的请求。

节点应该使用 LRU 策略来管理其路由中的条目请求表。

要在每个 Route 请求中保留的标识值的数量表条目，RequestTableIds，绝不能无限制，因为，在最坏的情况，当一个节点崩溃并重新启动时，第一个 RequestTableIds 路由发现重启后启动它可以似乎与网络中的其他节点重复。在另外，节点应该基于其初始标识值使用重

新启动后路由发现，在电池备份时钟上或其他持久性存储设备（如果有），以便提供帮助避免在成功发现新路线时出现任何可能的延迟重启后; 如果没有这样的初始识别值来源可用，重启后的节点应该以其初始为基础随机数的识别值。

4.无偿路线回复表

实现 DSR 记录的节点的 Gratuitous Route Reply Table 有关此节点发送的“免费”路由回复的信息自动路线缩短的一部分。a 节点在听到数据包时会返回无偿路由回复由某个节点发送，节点听到该数据包不是预期的下一跳节点，但后来被命名为该数据包中源路由的未扩展跳数; 节点无意中听到的数据包返回一个无偿的路由回复数据包的原始发件人，列出较短的路由（不是包括源路由的“跳过”的跳跃包）。节点使用其 Gratuitous Route Reply Table 来限制它产生无偿路线回复的速度它从无意中听到数据包到的同一节点的原始发送者触发无偿路线回复。

节点的 Gratuitous Route Reply Table 中的每个条目都包含以下字段：

- 此节点发起无偿的节点的地址路线回复。
- 此节点无意中听到数据包的节点的地址触发那个无偿的路由回复。
- 此条目在无偿路线之前的剩余时间回复表过期，应该被节点删除。

当节点无意中听到会触发无偿路由的数据包时如果节点中已存在相应的条目，则回复无偿路由回复表，那么节点应该不发送对该数据包的无偿路由回复。否则（即，如果没有相应的条目已经存在），节点应该创建一个新的在其免费路线回复表中输入以记录无偿路由回复，超时值为 GratReplyHoldoff。

5.网络接口队列和维护缓冲区

取决于诸如结构和组织等因素操作系统，协议栈实现，网络接口设备驱动程序和网络接口硬件，数据包正在传输可以以各种方式排队。例如，来自网络协议栈的传出数据包可能排在队列中操作系统或链路层，在传输之前网络接口。网络接口也可能提供用于分组的重传机制，例如在 IEEE 802.11 中作为路线维护的一部分，DSR 协议要求有限缓冲已传输的数据包尚未确定下一跳目的地的可达性。这里根据两个概念数据定义 DSR 的操作结合这种排队行为的结构。

实现 DSR 的节点的网络接口队列是输出来自网络协议栈的数据包队列等待由网络接口传输；例如，在 4.4BSD Unix 中网络协议栈实现，这个队列用于网络 interface 被表示为“struct ifqueue”。这个 queue 用于在网络接口所在的情况下保存数据包传输另一个数据包的过程。

实现 DSR 的节点的维护缓冲区是一个队列此节点发送的正在等待下一跳可达性的数据包确认作为路线维护的一部分。对于每个数据包维护缓冲区，一个节点维护一个计数重传和上次重传的时间。数据包是在第一次传输尝试后添加到维护缓冲区是。维护缓冲区的大小可能有限；添加时如果缓冲区大小为，则为维护缓冲区的新数据包不足以容纳新数据包，新数据包应该是默默地丢弃。如果，在 MaxMaintRexmt 尝试确认之后一些节点的下一跳可达性，没有收到确认，全部此节点的维护缓冲区中的数据包具有此下一跳目的地应该从维护缓冲区中删除。在这在这种情况下，节点也应该为此数据包发起路由错误以这种方式删除数据包的每个原始来源并且应该以这种方式挽救每个被移除的包它有另一条路由到该数据包的 IP 目的地址路由缓存。MaxMaintRexmt 的定义在概念上包括可能为链路上的数据包尝试的任何重新传输层或网络接口硬件内。超时值为用于确认请求的每次传输尝试取决于 Route 使用的确认机制的类型维护该尝试。

6.黑名单

使用 DSR 协议的节点通过网络连接时需要物理双向链路进行单播的接口传输时，节点必须保持黑名单。黑名单是一个表，由邻居节点地址索引，表示链接此节点与指定的邻居节点之间可能不存在双向的。节点将另一个节点的地址放在此列表中当它认为来自该其他节点的广播数据包到达时此节点，但两个节点之间的单播传输不是可能。例如，如果转发路由回复的节点发现如果下一跳不可达，则将下一跳放入节点的黑名单。

一旦节点发现它可以双向通信黑名单中列出的其中一个节点，它应该删除该节点来自黑名单。例如，如果节点 A 在其中列出了节点 B.黑名单，但在发送路由请求后，节点 A 听到 B 使用路由记录转发路由请求，指示该路由从 A 到 B 的广播成功，然后 A 应该删除该条目从其黑名单中获取节点 B.

节点必须将状态与其黑名单中列出的每个节点相关联，指定到该节点的链接的单向性是否是“可疑的”或“可能的”。每次不可达性肯定地，节点应该将状态设置为“可能”。在一些人没有得到肯定的不可达性之后一定的时间，国家应该恢复“有问题”。一个节点在合理的情况下，MAY 可以从其黑名单中过期节点的条目多少时间。

7.流程表

实现流状态扩展的节点必须实现 Flow 表或其他数据结构与外部行为一致在本节中描述。未实现流状态的节点扩展不应该实现流表。

流表记录有关数据包流的信息最近已被此节点发送或转发。表是由三元组索引（IP 源地址，IP 目的地址，流 ID），其中 Flow ID 是源指定的 16 位数字,流表中的每个条目都包含

以下字段：

- 沿此流的下一跳节点的 MAC 地址。
- 指示此节点上的传出网络接口的指示

  用于沿此流传输数据包。
- 沿此流的前一跳节点的 MAC 地址。
- 此节点上的网络接口的指示接收来自该前一跳节点的分组。
- 超时之后，流表中的此条目必须是删除。
- DSR 流状态中 Hop Count 字段的预期值收到的用于沿此字段转发的数据包的标头（for 与包含 DSR 流状态头的数据包一起使用）。
- 指示此流是否可用作默认值此节点发起的数据包的流量（默认的流 ID）流量必须是奇数）。
- 条目应该记录流的完整源路由。（不记录完整源路由的节点不能参与在自动路线缩短。）
- 条目可以包含记录此条目时间的字段最后使用。

必须在超时到期时删除该条目。

8.自动路线缩短表

实现流状态扩展的节点应该实现一个自动路由缩短表或其他数据结构一致与本节中描述的外部行为。一个节点没有实现流状态扩展应该不实现自动路线缩短表。

自动路线缩短表记录有关的信息收到自动路由缩短的数据包可能。该表由三元组（IP 源地址，IP）索引目的地址，流 ID）。自动路线中的每个条目缩短表包含（包标识符，跳数）列表该流程的配对。列表中的分组标识符可以是任何接收数据包的唯一标识符; 例如，对于 IPv4 数据包，来自数据包 IP 的以下字段的组合 header 可以用作数据包的唯一标识符：Source 地址，目的地址，标识，协议，片段偏移量和总长度。条目列表中的 Hop Count 是从 DSR Flow State 标头中的 Hop Count 字段复制收到已创建此表条目的数据包。任何包标识符应该在条目列表中最多出现一次，这个 list item 应该记录为此收到的最小 Hop Count 值数据包（如果无线信号强度或信噪比为接收到的数据包可用于 DSR 实现例如，一个节点，节点 MAY，可以在这个列表中记住接收数据包信号的最小跳数值强度或信噪比超过某个阈值）。

节点的自动路由缩短表中的空格可以是由节点上的任何本地算法动态管理。例如，为了限制用于存储表的内存量，任何可以随时删除现有条目和数据包的数量列入每个条目可能是有限的。但是，当回收空间时在表中，节点应该倾向于保留更多信息

流入表中而不是每个中列出的更多数据包在表格中输入，只要至少上市一些小可以在每个条目中保留分组数量。

9.默认流 ID 表

实现流状态扩展的节点必须实现 Default 流表或其他数据结构与外部一致本节中描述的行为。未实现流的节点状态扩展不应该实现默认流表。

对于每个（IP 源地址，IP 目标地址）对，a 节点转发数据包，节点必须记录：

- 看到的最大奇数流 ID 值。
- 转发的所有相应流的时间
- 当前的默认流 ID。
- 表示当前默认流 ID 是否为的标志有效。

如果节点删除此记录（IP 源地址，IP 目标地址）对，它还必须删除所有流表条目为那对。节点必须删除表条目，如果所有这些（IP 源地址，IP 目标地址）对的流量由此节点转发过期。

10.DSR 选项标头格式

动态源路由协议使用特殊标头携带可包含在任何现有 IP 中的控制信息包。数据包中的此 DSR 选项标头包含一个小的固定 - 大小的 4 字节部分，接着是零或更多 DSR 的序列带有可选信息的选项。序列的结束 DSR 选项标头中的 DSR 选项由总长度暗示 DSR 选项标题。

对于 IPv4，DSR 选项头必须紧跟 IP 数据包中的标头。（如果是 Hop-by-Hop 选项扩展标头，则为在 IPv6 中定义，为 IPv4 定义了 DSR 选项标题必须紧跟在 Hop-by-Hop 选项扩展之后标头，如果数据包中存在一个，否则必须立即按照 IP 标题。）

要将 DSR 选项标头添加到数据包，DSR 选项标头是在任何后续之前插入数据包的 IP 头之后标题，例如传统（例如，TCP 或 UDP）传输层头。具体而言，IP 头中的 Protocol 字段用于指示 DSR 选项标头跟在 IP 标头后面 DSR Options 标题中的 Next Header 字段用于指示以下是协议头的类型（例如传输层头）DSR 选项标题。

如果任何标头跟随数据包中的 DSR 选项标头，则为总计 DSR 选项标题的长度（以及总长度，组合长度）存在的所有 DSR 选项）必须是 4 个八位字节的倍数。这个要求保留了以下标题的对齐方式包。

第四章 具体算法实现

```
struct dsr_pkt *dsr_pkt_alloc(struct sk_buff *skb)?/动态分配 buff 内存
{
    struct dsr_pkt *dp;
    int dsr_opts_len = 0;

    dp = (struct dsr_pkt *)MALLOC(sizeof(struct dsr_pkt), GFP_ATOMIC);

    if (!dp)
        return NULL;
```

```
memset(dp, 0, sizeof(struct dsr_pkt));

if (skb) {
/*    skb_unlink(skb); */

      dp->skb = skb;

      dp->mac.raw = skb->mac.raw;
      dp->nh.iph = skb->nh.iph;

      dp->src.s_addr = skb->nh.iph->saddr;
      dp->dst.s_addr = skb->nh.iph->daddr;

      if (dp->nh.iph->protocol == IPPROTO_DSR) {
            struct dsr_opt_hdr *opth;
            int n;

            opth = (struct dsr_opt_hdr *)(dp->nh.raw + (dp->nh.iph->ihl << 2));
            dsr_opts_len = ntohs(opth->p_len) + DSR_OPT_HDR_LEN;

            if (!dsr_pkt_alloc_opts(dp, dsr_opts_len)) {
                  FREE(dp);
                  return NULL;
            }

            memcpy(dp->dh.raw, (char *)opth, dsr_opts_len);

            n = dsr_opt_parse(dp);
```

```
                DEBUG("Packet has %d DSR option(s)\n", n);
            }

            dp->payload = dp->nh.raw +
                (dp->nh.iph->ihl << 2) + dsr_opts_len;

            dp->payload_len = ntohs(dp->nh.iph->tot_len) -
                (dp->nh.iph->ihl << 2) - dsr_opts_len;

            if (dp->payload_len)
                dp->flags |= PKT_REQUEST_ACK;
        }
        return dp;
}

#endif
```

此函数可在接收和发送 pkt 时实现动态调节 buff 的大小。

dsr 路由流程：

nexthop_list_init(); 初始化下一跳列表，主要用来保存下一跳的信息，确定 ack 的请求机会

gw_list_init(); 初始化网关列表

routecache_init(); 初始化路由缓存

reqtable_init(); 初始化路由请求表，记录收到的路由请求的信息

rtsmbuffer_init(); 初始化重发缓冲区，保存发出数据中，需要 ack 回复的数据报的信息

sendbuffer_init(); 初始化发送缓冲区 ，临时存储 发送路由请求之后，受到路由应答请求之前，发送的数据报

dsr_stat_init(); 初始化统计信息

init_packet_queue(); 初始化用户空间队列

对全局变量 ipq_queue_t q 进行初始化，

注册一个处理 netfilter 队列的函数 netfilter_receive，一旦有数据进入队列就调用,做两件事情：

1.调用 ipq_enqueue 函数（收到的包和发出的包都由他加入到 q 队列中，进入的包有 dsr 头，而发出的包没有 dsr 头，所以在程序中使用了两次 switch 来吧接受的发送的 udp 包添加的 q 中）

发出的 hello 包，info 设置为空，根据不同的协议头，把元素加入到不同的 list 中

接受到的数据包，info 字段是由 netfileter 自己填写的

2.唤醒 dsr 进程。

startup_dsr(); 创建 DSR 进程，负责处理用户空间队列的数据报

注册 pre_route local_out gw_post_route3 个钩子 ,gw_post_route 为最后一个被调用的钩子

pre_route 钩子

1 如果发现数据包中 dsrhdr 的头部需要 ack 标志为 1，则直接发送一个 ack 回复，不进入用户空间排队。

2 如果是 ack 数据包，也就是下游节点返回的 ack 应答，然后进入 dsr_in_ack_op_handler 中进行处理，之前的判断已经没有必要了，不过还是按照从前的习惯保留了。其中在 dsr_in_ack_op_handler 中调用 rtsmbuffer_ack_del 函数,这个函数用来根据 ack 回复时间动态调整 ack 超时，这是和从前 dsr 最大的区别:

如果是数据包，则直接去掉 dsr 头，返回 netfilter

其他类型的数据包，则加入用户队列 (返回 NF_QUEUE)

加入队列的数据包，有 netfitlerreceive 调用 packet_enqueue 来完成，并触发 dsr 进程，然后再由 packet_in 和 packet_out 进行处理

dsr_local_out 流程

如果是自己给自己的，则直接返回

如果是发往外网段的直接返回

其他的数据报全部返回到用户空间队列排队。加入队列的数据包，有 netfitlerreceive 调用 packet_enqueue 来完成，并触发 dsr 进程，然后再由 packet_in 和 packet_out 进行处理

dsr 进程流程

循环从用户空间队列中取出元素 （iqp_dequeue）

{

进入的数据报 ,调用 packet_in

发出的数据报,调用 packet_out

}

packet_in 的流程

循环处理 dsr 头部字段

如果是 DSR_SRC_OPT :调用 dsr_src_opt_handler ，

如果 DSR_IN_OPT，调用 dsr_in_src_opt_handler 处理：

如果 dsrhdr 标志位中需要 ack 为 1,则发送一个 ack

如果 DSR_FORWARD_OPT，调用 dsr_forward_src_opt_handler

1 重新设置接受地址为下一跳地址，源地质为本节点地址；

2 重新设置 nexthop_info_entry 中下一跳的 ack 请求时间，即：

if_in_nexthop_list 判断转发的包的目的地址是否在下一跳得列表中。

Get_nexthop_natime 来获得请求下一个跳节点 ack 的时间。从 nexthop_info_entry 中得到最近一次从下一收到的 ack 时间。改变 Nexthop_info_entry 中的标志位，需要 ack ，计算下一次需要 ack 的时间；

3 修改 ip 头目的地址为本节点地址，源地址为本节点地址

如果是 DSR_REQ_OPT ,调用 dsr_req_opt_handler

如果是 DSR_IN_OPT ，则调用 dsr_in_req_opt_handler，

1 首先处理路由信息，添加从本节点到路由请求发起源的路由信息。

2 再次判断是否为网关请求，然后 make_and_send_dsr_reply 回复一个网关应答，否则回复一般的请求 同样是使用 make_and_send_dsr_reply

3 修改 ip 头目的地址为本节点地址

如果是 DSR_FORWARD_OPT dsr_forward_req_opt_handler

1 判断是否收到过这个请求，如果收到过，则返回，否则继续

2 处理路由信息，添加从本节点到路由请求发起源的路由信息

3 然后针对网关修改，首先查找本的路由网关列表，没有则转发网关请求：iph->daddr= BCAST;

4 如果本节点网关列表不为空，则还是 make_and_send_dsr_reply 来发送回复，将本地有的网关路由发送给请求节。

如果是 DSR_REP_OPT ，调用 dsr_rep_opt_handler

如果是 DSR_IN_OPT, 则 dsr_in_rep_opt_handler

首先先把添加本地路由信息，然后添加网关列表，然后处理发送缓冲区的数据，这个缓冲区的数据，是在 make_dsr_source_route_option 中调用 sendbuffer_insert 添加的。

如果是 DSR_FORWARD_OPT, 则调用 dsr_forward_rep_opt_handler ,

首先将 reply 信息中自己需要的路由信息提取，添加本地路由信息，如果是网关发现应答，然后则提取网关路由，并添加网关列表

如果是 DSR_ERR_OPT ，则 dsr_err_opt_handler

如果 DSR_IN_OP 则 dsr_in_err_opt_handler

如果 DSR_FORWARD_OPT dsr_forward_err_opt_handler

循环结束

如果数据包是给自己的，则从 dsr 头部恢复真正的源地址和目的地址（针对网关设计的），而后去掉 dsr 头部，然后返回 nf_accept

否则，则直接由自己转发，返回 nf_stolen

重新注入 netfilter 架构：nf_reinject（）

paket_out 流程：

调用了 make_and_send_dsr_data_packet 进行处理具体过程：

1 如果 hoplist 是空 则调用 make_dsr_source_route_option ，在这个函数中，首先比较目的地址是否为本网段，如果不是本网段则需要查找本地网关列表

如果没有网关信息，则发起网关发现：gateway_discovery()

否则检查到网关/内网地址由，如果没有路由，则发起路由请求，将发送的数据暂时保存在发送缓冲区中，【在收到路由请求应答之后来发送】

总结一下 make_dsr_sour_route_option 的功能，就是检查网关信息和路由信息，没有相关信息则发起路由请求，把要发送的数据存到发送缓冲区中

2 如果 hoplist 不为空 则 调用 make_dsr_source_route_opt 直接将 hoplist 复制给 scropt 的 hoplist.

3 ,填写 dsrhdr 的头部信息

4.获取下一跳的地址 填写 ip 头的目的地址

5．对 nexthop_info_entry 中的 ack 请求时间进行修改,并根据条件修改 dsr 需要 dsr 头部的需要 ack 标志位。

6．dsr_merge_opt_two 合并 dsr 头和源路由选项，并发送 insert_dsr_stuff_and_send

在 insert_dsr_stuff_and_send 中 又调用 dsr_send_packet，而后在 dsr_send_packet 中

调用了 rtsmbuffer_insert 来插入发送缓冲区一个记录

回到 packet_out，然后返回一个 NF_STOLEN

网关节点的钩子调用处理过程

说明几个钩子

dsr 的 nat

dsr_pre_route 优先级最高 pre_route 优先级低于 dsr_pre_route

pre_route ，本在发出之前的 localout 优先级高于 gw_local_out

gw_local_out 上线之前

第一部分 ，从网关受到发往外网段的数据报：

首先调用 dsr_pre_route，进行各种头部处理，然后恢复 ip 头，交付给 netfilter.

这时候 nat 的 pre_route 被调用，nat 负责转换原地质为网关地址，记录内外对应的 ip 信息和段口号，返回 netfilter，然后在发出是掉用 localout 转换目标地址，返回给 netfilter,

此时 gw_local_out 由于也是挂在 local_out 处，优先级低，所以在 nat 的 localout 结束后会被调用，察看是不是由内网发给内网的，不是则直接返回 nf_accept,由此一个外发的数据报处理完毕。

第二部分 ，从外网发给内网的

首先由 dsr_pre_route 处里，由于外网进入的是原始的数据报，所以直接返回 ，再次有 nat 的 preroute 捕获，进行原地址的变换，转为网关地质，经过内外对应的 iptable，再 local-out 出 转换目标地址为内网结点地址，然后 gw_local_out 再次截获，发现是由外网发到内网段的，则进行处理调用 make_and_send_dsr_data_packet,转为 dsr 协议可以识别的包，发送出去。

源代码包括：

cc(H)：DSR 代理类。主要状态机处理路由。重要变量；ID 类型中的 netid、mac_id。(ip 和 mac 地址)，它们都由 tcl 命令初始化，以设置 initla valie，逗号是“addr”和“mac_addr”

HDPR_TC(h)--定义 HDPR_Dos 先生。

请求 table.cc(H)

path.h(Cc)：path 类。首先，定义 structID，它有一个无符号的长 addr、id_type 的枚举和时间戳 t。然后在 PATH 类中，ID[]是路径的关键成员，运算符[]是 de 罚款以返回 ID 数组的元素。因此，每当 SRPacket.path[n]将返回到 ID[n]的引用时。其他成员变量包括 cur_index、len、

h：只需定义 SRPacket 类，它将 HDR_Sr 封装为一个完整的数据包。SRPacket 构造有两个参数，一个普通数据包和一个 SR(源路由)报头(路径变量)。坏处 “路径”类的卡车司机从 NS 源路由头的位数创建路径。SR 包的另外两个变量是“DST”和“src”IP 地址。

路由数据结构
发现 SRH->ADR 和 P.路由是两种不同的结构。SRH()总是伴随着数据包。但是，当 dsr 代理接收到数据包时，它将

这会生成一个“p”，它经常被所有其他函数使用。请记住，p 不与离开 DSR 代理的数据包一起运行。在数据包被发送出代理之前，另一条语句将 我被用来更新 sendOutPacke 中的“SRH”

另外，TAP()条目也为其使用生成一个 p。但是，代理“xmitFails”的另一个入口点直接使用 SRH()。

特殊的 TCL 接口。

除非其他路由协议，ns-2.1b9a 有一个特殊的节点类型命名为“SRNodeNew”。从 ns-lib.tcl 的那些例行公事中。我们可以看到它的特点。此外，Mobiloty/dsr.tc 中还有一个 tcl 文件。 我也是亲戚。

DSR 信令数据包简介：

路线-请求。包本身是一个具有唯一目标地址的第三层数据包，但是 MAC_广播(在公共报头的 Next_HOP()中标记)

路由-回复：第二层和第三层的单播。

路线错误。第二层和第三层的单播。当 TX_FILE 在下层发生时生成。

DSR 代理的入境点：

首先，与正常情况一样，recv()函数意味着一个具有地址的数据包到此节点或来自上目标。

xmitFailed()这是 MAC 传输失败时的回调函数。基于此机会，生成路由错误消息。

拍()这是一个隐藏的条目，当你打开乱交。窥探路线缩短路线。

基本职能：

recv()，重装包的条目。取决于数据包的 IP 地址和 SR(源路由)报头，调用不同的函数来处理它。如下图所示：

路由-应答、路由请求消息需要用 handlePacketReceipt()函数编写的特殊处理例程。注意，如果 RRequest 消息到达目的地，则接收程序应该 发送一个 RReply 消息，这是由一个名为 backnSrcRouteToRequestor(P)的函数完成的，这个函数在 handlePacketReceipt()中被调用。否则，如果路由请求和路由错误

或者不是命中注定的，对于路由请求，函数 handleRouteRequest()被调用

handlePacketReceipt()一个信号到达了目的地。有两种情况：

如果是路由请求而“未处理”，则发回路由应答，pkt“p”在此函数中是伪造的，并返回 SrcRouteToRequestor(P)。

如果是路由-应答，呼叫功能可接受 RouteReply(P)；

handleRouteRequest()：从 2.27 版本中，我们看到了一些未使用的代码，但可能正在为将来的版本开发。它包括更紧密的 mac 路由合作，例如邻居标识(Is_Nei)。 和信道状态(air_time_free()。

基本上，这个功能有三个分支：

已经处理完毕，由函数 crereRouteRequestp(P)检查。

有一个缓存的路由，由 responyFromRouteCache(P) 完成，而“缓存路由”是由 dsragent_RER_FROM_CACHE_ON_PERATING 标志启用的。

在路由中添加 p.route.附录 ToPath(Net_Id)；和 sendOutPacketWithRoad(p，false)；

手向前。在源路由中将数据包转发到下一个主机，并酌情进行窥探。因此，路由回复消息并不被视为例外。这是一个正常的包与 sr 报头和窥探 d 在这个节点上。窥探是由标志“dsragent_noop_source_path”启用的。

handleForwarding 是 dcallin“handleDefaultForwarding”，用于对 DSR 规则执行一些简单的操作。最后，数据包将由 sendOutPacketWith 路由发送

sendOutPacketWithRouting：该函数用于发送数据包、获取数据包并将其发送出去，数据包必须有一个路由。返回值不是很有意义。如果 RESTERY 为真，则重置 p 在使用它之前，如果 RESTERY 为 false，则调用方希望我们使用当前设置的索引设置的路径。基本上，设置 CMN 报头的故障回调函数和数据

CMN 报头的下一跳被设置为 DSR 头文件中下一跳的加号。地址类型也被设置。

将 SR 标头中的指针移到下一个(增加 1)。

实际上，第三个操作在实际的 DSR 实现中是无效的。若要在错误处理中消除此影响，我们最好先找到当前的 ip 地址，并在 sr h 中定位该地址的位置。 伊迪尔。另外两个操作也是无效的，因为在实际的数据包中没有通用的标头。对于公共报头中的“Next_HOP()”，DSR 只对那些没有有效 SR 的数据包使用 标头，请参阅 recv()。

back nSrcRouteToRequestor()；此函数

xmitFails()：当公共标头->xmit_Failure_指向回调函数时，当无法传递数据包时，将使用回调函数，并最终生成路由错误 MES 圣人。SR 头 Curr() 中始终有一个指针。(请参阅 MANET-IETF-DSR 草案，DSR 源路由选项中没有这样的指针，但有一个“段左”字段来指示多少段)。 节点仍需访问才能到达目的地。)因此，需要进行创新，将“SRH->cur_addr()”重新解释为失败发生路径中位置的索引号。所以，当 并非所有路径上的节点都处理 SR 报头，我们需要从 SRH 中找到该节点的 IP 地址。并将该索引设置为 cur_addr()；

processBrokenRouteError(P)；这应该是主 recv()条目下的另一个分支。它给出了当路由错误消息被重述或听到(窥探)时该做什么。Snoop 的意思是消息被发送了 到了另一个节点，但它是我自己传递的，所以我听到了。

TAP(ConstPacket*Packet)：这是 DSR 的另一个入口点。如果 dsragent_use_抽头标志为真。mac

在杂乱模式下工作，如果存在 它的头部是高级的。

我们可以处理的最长路由定义为：定义 MAX_SR_Len 16/我们可以处理的最长源路由

在 IP 层以下 xmit_Failure 的可能原因：

ARP 失效

接口队列已满

mac 传输失败(超过重试限制)

DSR 计划选项：

在 dsragent.cc 的开头，它定义了一些选项的许多 bool 选择器，如：

关于流动状态：

它也是可取的禁用流状态的东西。它使 DSR 代码变得混乱。流动状态不是一个自然的想法。

关于包打捞

目前尚不清楚如何在第三层中完全禁用数据包重传。甚至更改 dsragent.cc 中的三个表达式。

用_cache=false(从 true)打捞

救助请求=0(从 1)

救助次数=0(从 15)

跟踪文件仍然显示路由代理再次发送不可交付的数据包。请参见 xmitFail()函数。我想，也有必要禁用“GOD”。

总 结

这次网络协议我们两个人合作完成了 DSR 协议的大部分分析，短短半个学期，我们组两个人临时组成了一个团队来理解这个协议，用时六周，当然这离不开覃老师在课上的详实的讲解，整个大作业期间，我们通过各种方式查阅资料，争取把遇到的代码都能看懂，不懂就向身边的同学请教，然后把协议展示出来。通过对代码的分析，我们发现：DSR 协议的优缺点

优点:

采用源路由机制、避免了路由环路。

它是一种按需路由协议、只有当两个节点间进行通信时，才会缓存路由纪录，因此相对主动路由来说，减小了路由维护的开销。

通过采用路由缓存技术，减少路由请求信息对信道的占用

缺点:

随着路经跳数的增加，分组头长度线性增加、开销大

路由请求分组 RREQ 采用洪泛发向全网扩散，导致网络负荷大

来自邻居节点的 RREQ 分组在某个节点可能会发生碰撞，解决办法是：在发送 RREQ 分组时引入随机时延

当源节点发送路由请求分组 RREQ 时，可能会收到多个节点缓存的到达目的节点的路由信息，引起竞争。解决办法：若某节点听到其它节点发出的 RREQ 分组中路由信息含有较少跳数，此节点停止发送。

当源节点发送路由请求分组 RREQ 时，可能会收到多个节点缓存的到达目的节点的路由信息，但有些路由信息可能是过时的。解决办法：引入定时器、链路断的情况应进行全网洪泛。

参考文献

[1] DSR RFC 文档